OpenMp

OMP Data Scoping

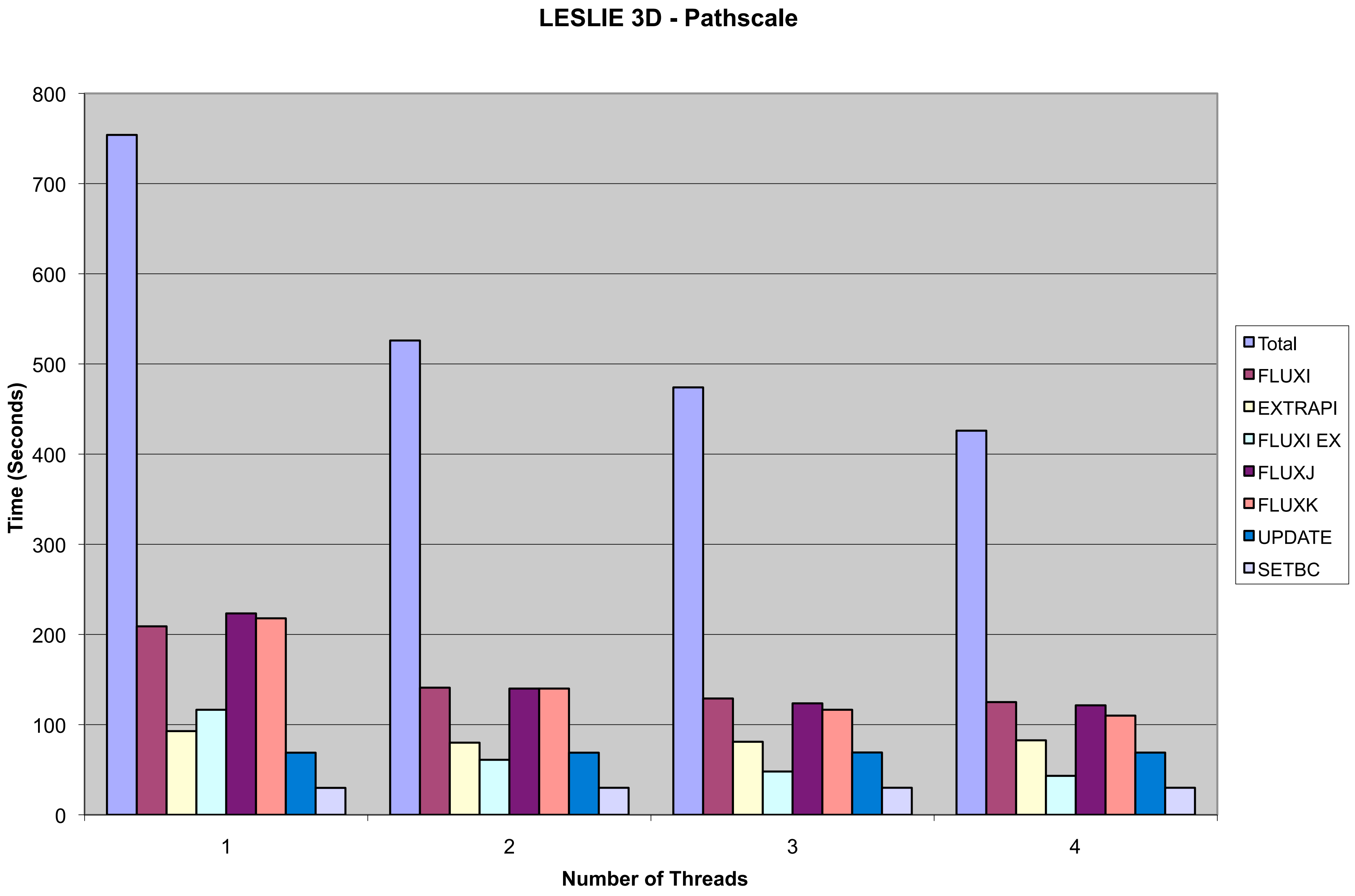
- Any variable that is invariant with respect to the parallel DO loop is a scalar quantity and if it is set in the parallel DO loop and then used it has to be private
- Any scalar quantity that is used and then set in the parallel DO Loop it gives rise to loop carried data recursion
- All Arrays referenced by the parallel loop index are shared
- Variables down the call chain revert to Fortran scoping rules
  - All COMMON, MODULE and SAVE statements have to be shared
  - All Automatic and allocated arrays have to be private
  - Data scoping down the call chain is EXTREMELY DIFFICULT

LESLIE3D OMP directives

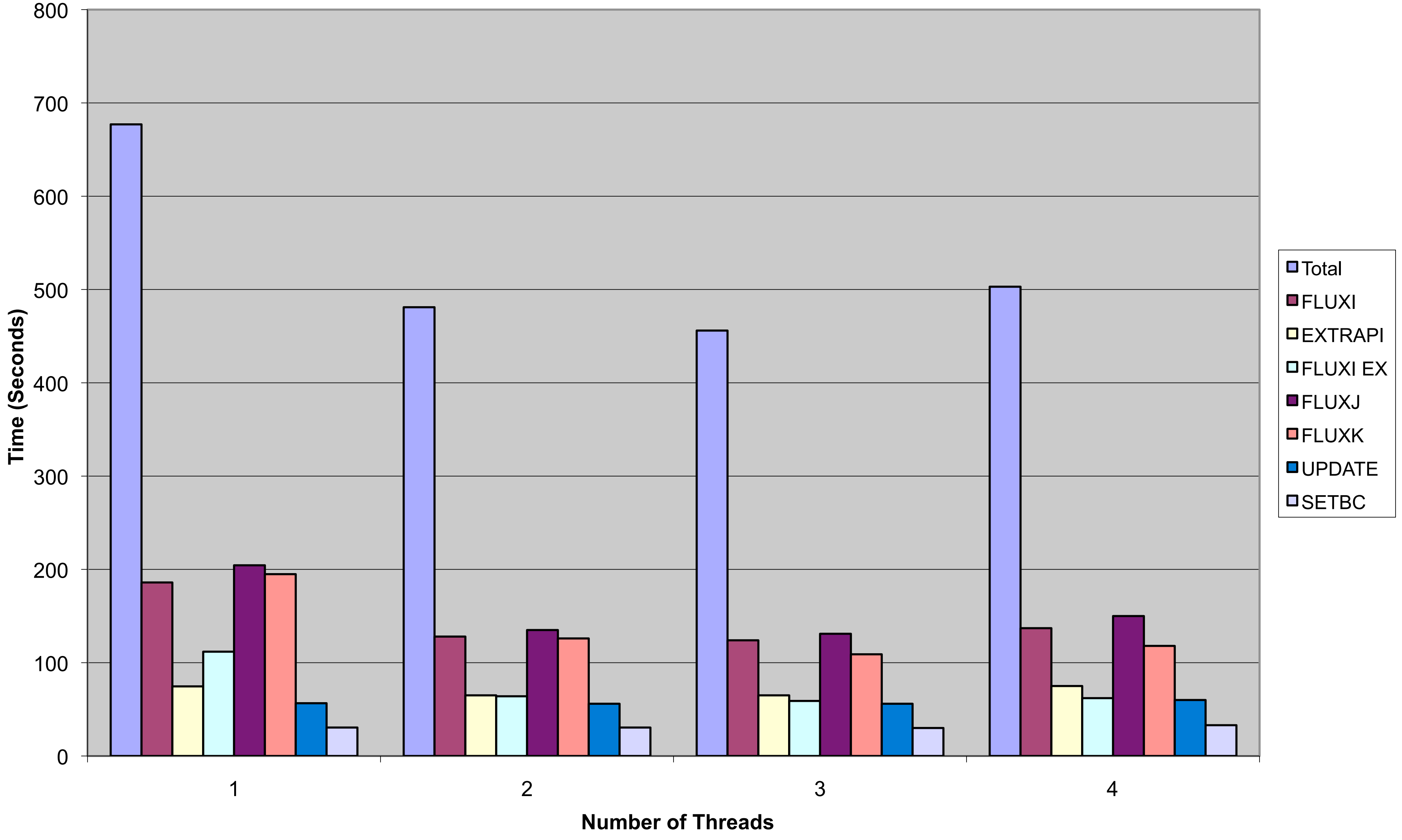
```
C$OMP PARALLELDO
C$OMP&SHARED (I1,CMU,GDIFFAC,DXI,RDENG,K2,DZI,KK,J2,CNUK,IBDD,XAREA)
C$OMP&SHARED(K1,I2,DYI,J1,DTVOL,JMAX,DELTA,IADD,N,DCDX,DVDZ,CPK,FSI)
C$OMP&SHARED(HFK,DWDY,QAV,WAV,CONC,PAV,EK,DU,VAV,HF,UAV,W,V,U,T,Q,H)
C$OMP&PRIVATE (RHEV,RHEK,TXX,HK,ABD,CPAV,TEMP,SXX,SGSXX,WAVE,RLMBDA)
C$OMP&PRIVATE (SGSEX,RHAVE,QSPI,VAVE,TZX,TXZ,RK,SGSXY,ICD,II,QSP,QX)
C$OMP&PRIVATE (UAVE,SYY,SXZ,NS,RD,SGSXZ,TYX,TXY,EKAVE,IBD,SXY,DIV,SZZ)
C$OMP&PRIVATE (L,J,I,QS,DVDY,DVDX,EKCOEF,DUDZ,DUDY,DUDX,DHDX,QDIFFX)
C$OMP&PRIVATE (DKDX,RMU,DTDX,DWDZ,DWDX,K)
      DO K = K1,K2
        DO J = J1,J2

          ABD = DBLE(IBDD)
!
! EULER STUFF
!
         DO I = I1,I2
            QS(I) = UAV(I,J,K) * XAREA
          END DO

          IF(NSCHEME .EQ. 2) THEN
            DO I = I1,I2
DO I = I1,I2
    DUDX(I) = DXI * (U(I+1,J,K) - U(I,J,K))
    DVDX(I) = DXI * (V(I+1,J,K) - V(I,J,K))
    DWDX(I) = DXI * (W(I+1,J,K) - W(I,J,K))
    DTDX(I) = DXI * (T(I+1,J,K) - T(I,J,K))
 ENDDO

 DO I = I1, I2
    DUDY(I) = DYI * (UAV(I,J+1,K) - UAV(I,J-1,K)) * 0.5D0
    DVDY(I) = DYI * (VAV(I,J+1,K) - VAV(I,J-1,K)) * 0.5D0
    DWDY(I) = DYI * (WAV(I,J+1,K) - WAV(I,J-1,K)) * 0.5D0
 ENDDO
```

```
  DO I = I1,I2
  FSI(I,2) = FSI(I,2) + SGSXX * XAREA
  FSI(I,3) = FSI(I,3) + SGSXY * XAREA
  FSI(I,4) = FSI(I,4) + SGSXZ * XAREA
  FSI(I,5) = FSI(I,5) + SGSEX * XAREA
END DO

DO I = I1,I2
 RDENG = EKCOEF(I) + RMU(I) / PRANDLT
  FSI(I,7) = FSI(I,7) - RDENG * DKDX(I) * XAREA
END DO

  DO NS = 1,NSPECI
    DO I = I1,I2
      FSI(I,7+NS) = FSI(I,7+NS) -
>               EKCOEF(I) * DCDX(I,NS) * XAREA
    END DO
  END DO
```

```
      IF(ISGSK .EQ. 1) THEN
       DO I = I1, I2
          RHAVE  = 0.5D0 * (Q(I,J,K,1,N) + Q(I,J+1,K,1,N))
          EKAVE  = 0.5D0 * (EK(I,J,K) + EK(I,J+1,K))
          EKCOEF(I) = RHAVE * CNUK * SQRT(EKAVE) * DELTA
          RHEV   = 2.0D0 * EKCOEF(I)
!           RDENG  = CENAVE * EKCOEF(I)
          RHEK   = TWO3 * RHAVE * EKAVE

          SXX = DUDX(I)
          SYY = DVDY(I)
          SZZ = DWDZ(I)
          SXY = 0.5D0 * (DUDY(I) + DVDX(I))
          SYZ = 0.5D0 * (DVDZ(I) + DWDY(I))

          DIV = (SXX + SYY + SZZ) * THIRD

          SGSYY = - RHEV * (SYY - DIV) + RHEK
          SGSXY = - RHEV * SXY
          SGSYZ = - RHEV * SYZ

          SGSEY = - RDENG * DHDY(I)

          FSJ(I,J,2) = FSJ(I,J,2) + SGSXY * YAREA
          FSJ(I,J,3) = FSJ(I,J,3) + SGSYY * YAREA
          FSJ(I,J,4) = FSJ(I,J,4) + SGSYZ * YAREA
          FSJ(I,J,5) = FSJ(I,J,5) + SGSEY
     END DO
```


LESLIE 3D - Pathscale
Time (Seconds)
800
700
600
500
400
300
200
100
0
1
2
3
4
Number of Threads
Total
FLUXI
EXTRAPI
FLUXI EX
FLUXJ
FLUXK
UPDATE
SETBC

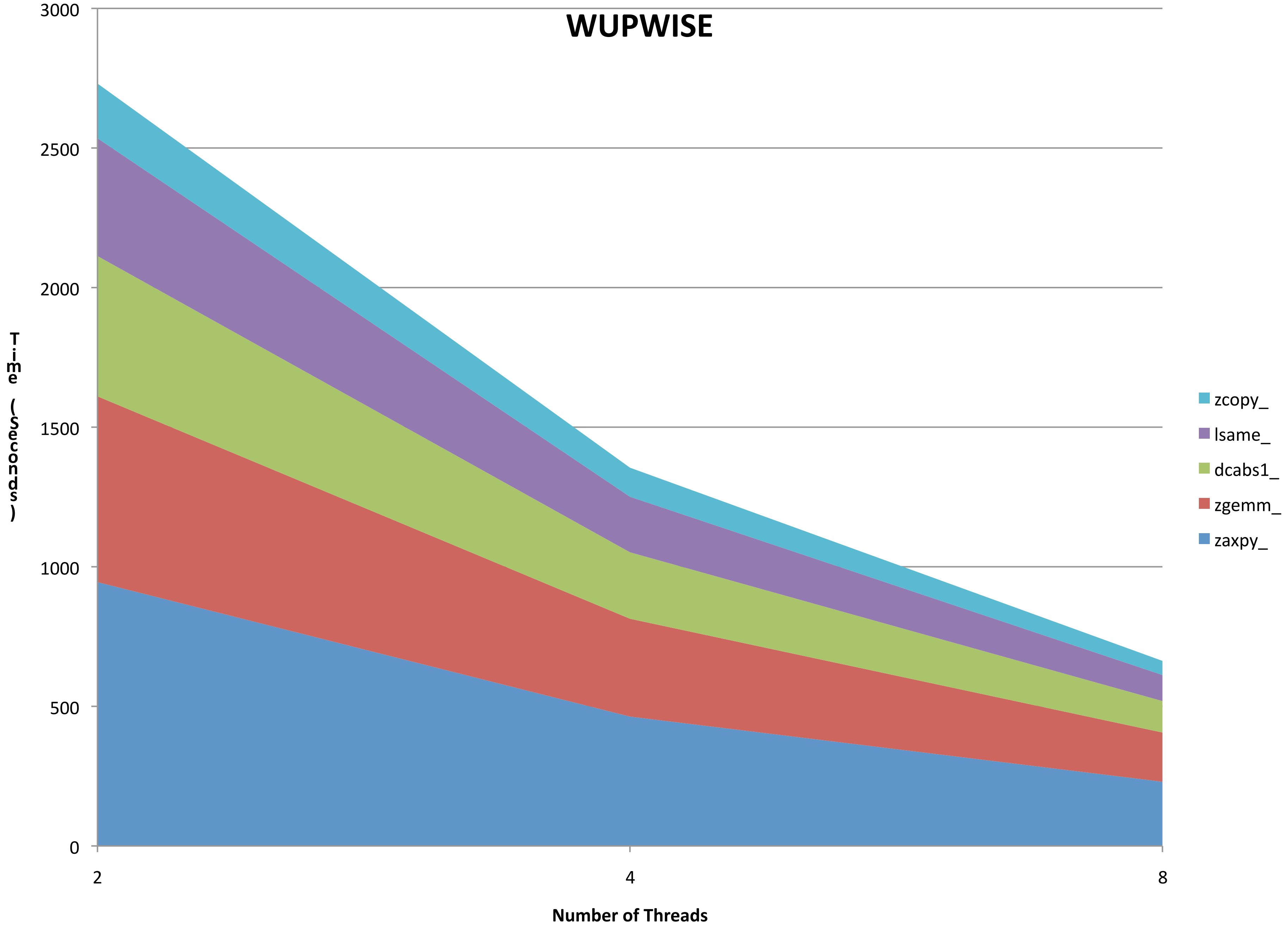
leslie3d - PGI



Performance Considerations

- Granularity of Computation
  - The parallel region should be as large, in number of operations, as possible
- Load Balancing
  - The work should be distributed evenly across the threads working the OpenMP region

SPEC OpenMP Routines

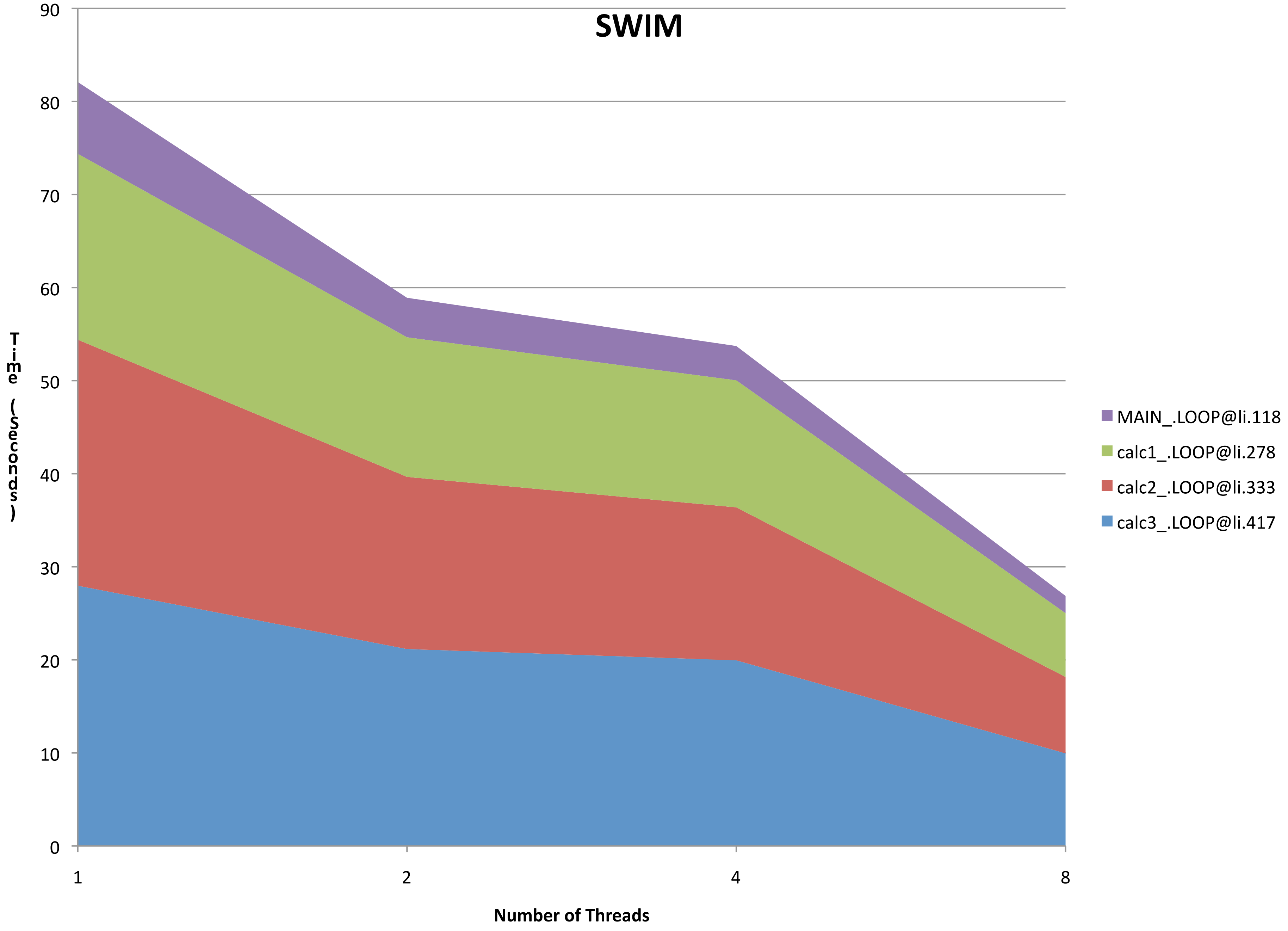

WUPWISE
3000
2500
2000
1500
1000
500
0
Time (Seconds)
2
4
8
Number of Threads
zcopy_
lsame_
dcabs1_
zgemm_
zaxpy_

Major OMP Loop in WUPWISE

```
C$OMP PARALLEL
C$OMP+        PRIVATE (AUX1, AUX2, AUX3),
C$OMP+        PRIVATE (I, IM, IP, J, JM, JP, K, KM, KP, L, LM, LP),
C$OMP+        SHARED (N1, N2, N3, N4, RESULT, U, X)
C$OMP DO
      DO 100 JKL = 0, N2 * N3 * N4 - 1
       L = MOD (JKL / (N2 * N3), N4) + 1
       LP=MOD(L,N4)+1
        K = MOD (JKL / N2, N3) + 1
        KP=MOD(K,N3)+1
         J = MOD (JKL, N2) + 1
         JP=MOD(J,N2)+1
         DO 100 I=(MOD(J+K+L+1,2)+1),N1,2
           IP=MOD(I,N1)+1
           CALL GAMMUL(1,0,X(1,(IP+1)/2,J,K,L),AUX1)
           CALL SU3MUL(U(1,1,1,I,J,K,L),'N',AUX1,AUX3)
           CALL GAMMUL(2,0,X(1,(I+1)/2,JP,K,L),AUX1)
           CALL SU3MUL(U(1,1,2,I,J,K,L),'N',AUX1,AUX2)
           CALL ZAXPY(12,ONE,AUX2,1,AUX3,1)
           CALL GAMMUL(3,0,X(1,(I+1)/2,J,KP,L),AUX1)
           CALL SU3MUL(U(1,1,3,I,J,K,L),'N',AUX1,AUX2)
           CALL ZAXPY(12,ONE,AUX2,1,AUX3,1)
           CALL GAMMUL(4,0,X(1,(I+1)/2,J,K,LP),AUX1)
           CALL SU3MUL(U(1,1,4,I,J,K,L),'N',AUX1,AUX2)
           CALL ZAXPY(12,ONE,AUX2,1,AUX3,1)
           CALL ZCOPY(12,AUX3,1,RESULT(1,(I+1)/2,J,K,L),1)
 100  CONTINUE
C$OMP END DO
```

Performance is Excellent

- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is very large
- Load Balance of the computation is good
- The computation that is parallelized is not memory bandwidth limited


SWIM
Time (Seconds)
90
80
70
60
50
40
30
20
10
0
1
2
4
8
Number of Threads
MAIN_.LOOP@li.118
calc1_.LOOP@li.278
calc2_.LOOP@li.333
calc3_.LOOP@li.417

Major OMP Loop in SWIM

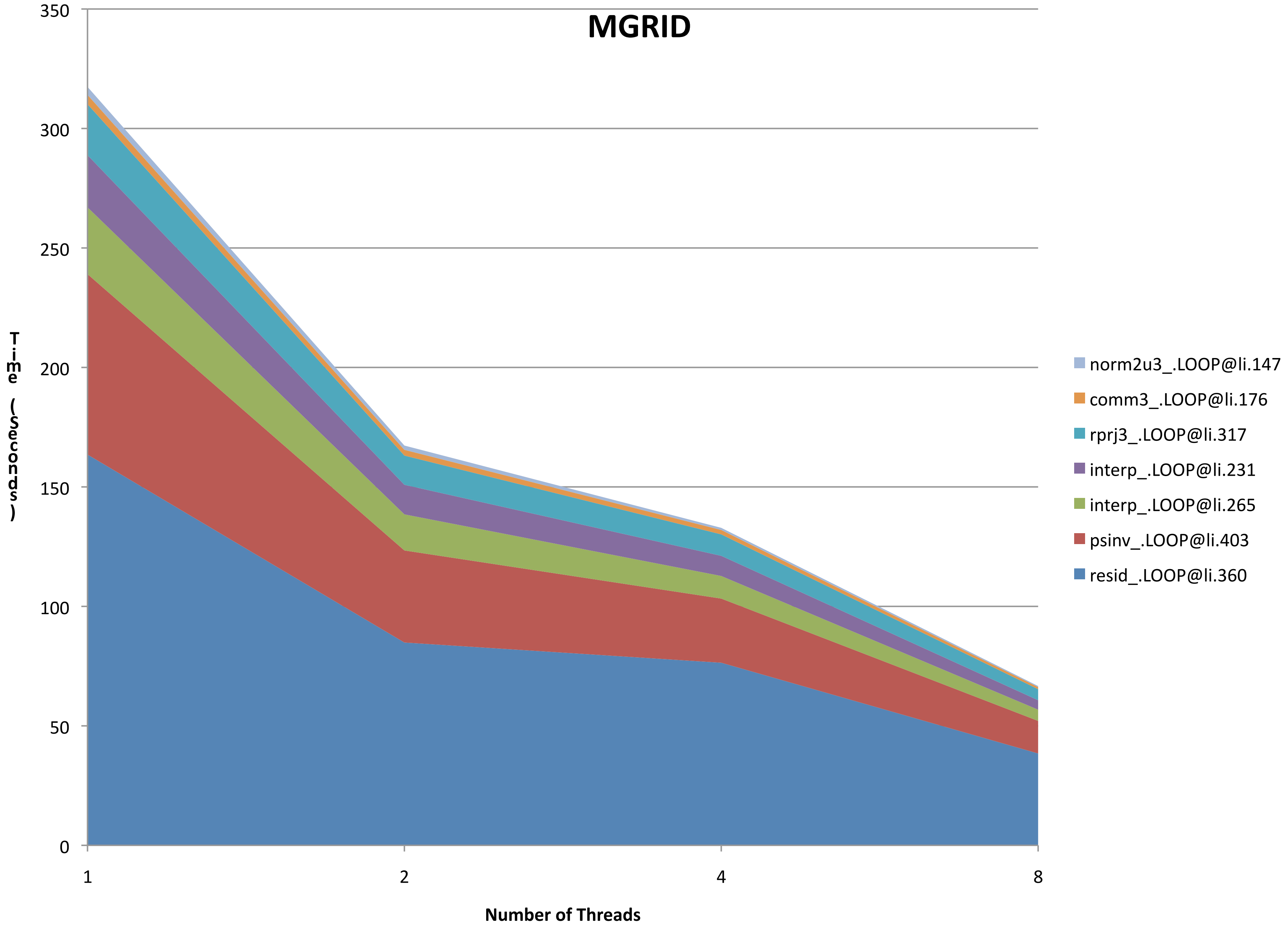
```
!$OMP PARALLEL DO
      DO 100 J=1,N
      DO 100 I=1,M
      CU(I+1,J) = .5D0*(P(I+1,J)+P(I,J))*U(I+1,J)
      CV(I,J+1) = .5D0*(P(I,J+1)+P(I,J))*V(I,J+1)
      Z(I+1,J+1) = (FSDX*(V(I+1,J+1)-V(I,J+1))-FSDY*(U(I+1,J+1)
     1          -U(I+1,J)))/(P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
      H(I,J) = P(I,J)+.25D0*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
     1               +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
  100 CONTINUE
```

Some Craypat Statistics

Time%	20.5%		
Time	18.120602		
Calls	120		
PAPI_L1_DCM	232.754M/sec	4175600353 misses	
PAPI_TLB_DM	1.334M/sec	23925543 misses	
PAPI_L1_DCA	871.323M/sec	15631527182 refs	
PAPI_FP_OPS	1932.840M/sec	34675154640 ops	
User time (approx)	17.940 secs	41262000000 cycles	99.0%Time
Average Time per Call	0.151005 sec		
CrayPat Overhead : Time	0.0%		
HW FP Ops / User time	1932.840M/sec	34675154640 ops	21.0%peak(DP)
HW FP Ops / WCT	1913.576M/sec		
Computational intensity	0.84 ops/cycle	2.22 ops/ref	
MFLOPS (aggregate)	1932.84M/sec		
TLB utilization	653.34 refs/miss	1.276 avg uses	
D1 cache hit,miss ratios	73.3% hits	26.7% misses	
D1 cache utilization (M)	3.74 refs/miss	0.468 avg uses	

Performance is Poor

- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is very large
- Load Balance of the computation is good
- The computation that is parallelized is memory bandwidth limited
  - Cache utilization is very bad so this is heavily dependent on memory loads/stores


MGRID
350
300
250
200
150
100
50
0
Time (Seconds)
1
2
4
8
Number of Threads
norm2u3_.LOOP@li.147
comm3_.LOOP@li.176
rprj3_.LOOP@li.317
interp_.LOOP@li.231
interp_.LOOP@li.265
psinv_.LOOP@li.403
resid_.LOOP@li.360

Major OMP Loop in SWIM

MGRID is a one of the NASA Parallel Benchmarks, it applies a multi-grid solver on a three dimensional grid. From the graph we once again see performance illustrative of a memory bound application.

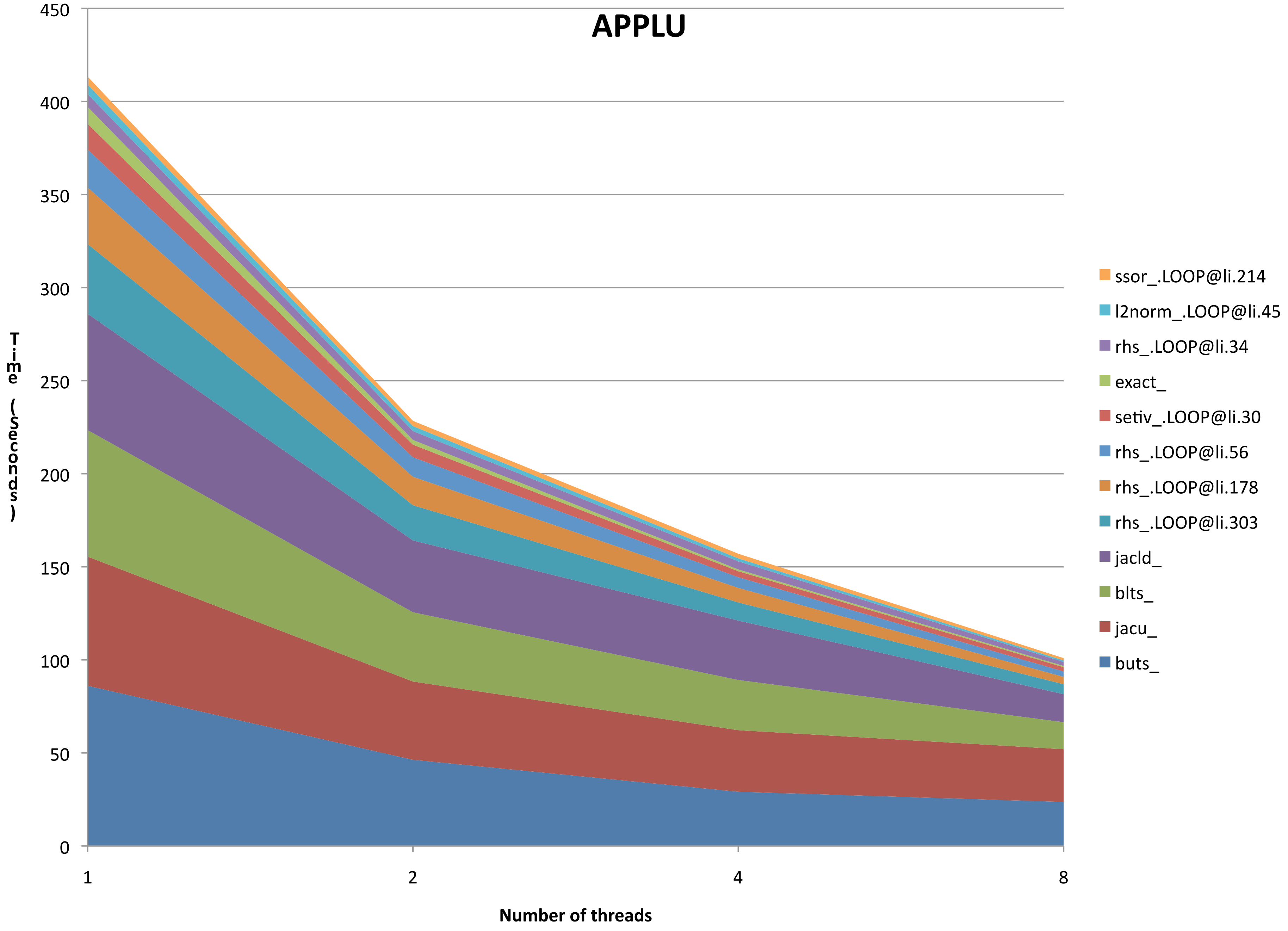
In this case the Level 1 cache utilization is very good; however, the Level 2 cache utilization is very poor. Following are the derived metrics from hardware counters for Level 1 and 2 cache on the resid routine.

D1 cache hit,miss ratio	96.3% hits	3.7% misses
D2 cache hit,miss ratio	9.2% hits	90.8% misses

The important lesson is that poor cache utilization will steal OpenMP scalability, simply because it increases the reliance on memory bandwidth which is the rarest of all commodities on the node.

Performance is Poor

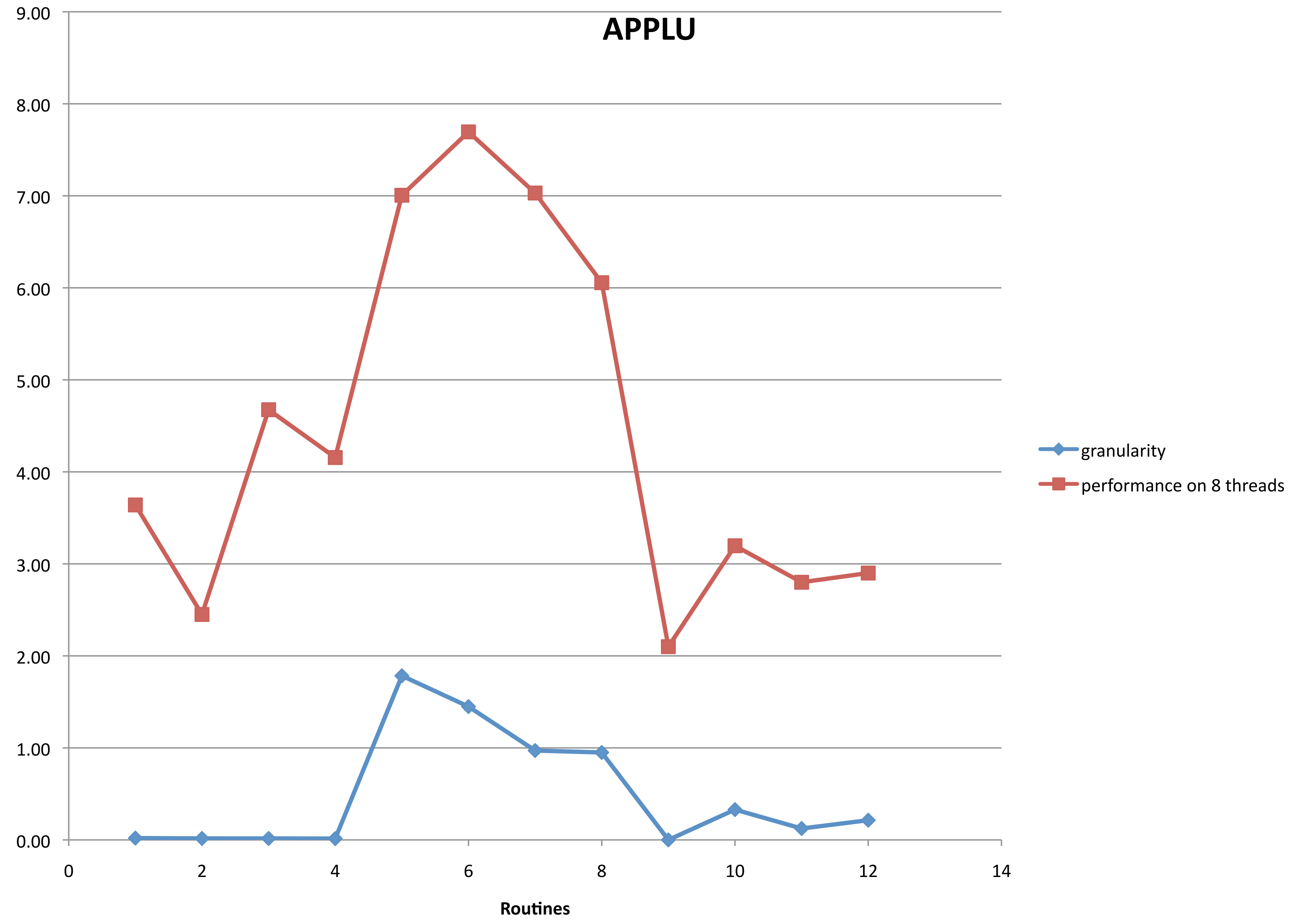
- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is very large
- Load Balance of the computation is good
- The computation that is parallelized is very memory bandwidth limited
  - Cache utilization is very bad so this is heavily dependent on memory loads/stores


APPLU
Time (Seconds)
450
400
350
300
250
200
150
100
50
0
1
2
4
8
Number of threads
ssor_.LOOP@li.214
l2norm_.LOOP@li.45
rhs_.LOOP@li.34
exact_
setiv_.LOOP@li.30
rhs_.LOOP@li.56
rhs_.LOOP@li.178
rhs_.LOOP@li.303
jacld_
blts_
jacu_
buts_

APPLU is another NASA parallel benchmark which performs the solution of five coupled nonlinear PDE's, on a 3-dimensional logically structured grid, using an implicit psuedo-time marching scheme, based on two-factor approximate factorization of the sparse Jacobian matrix. The following chart shows that some routines scale very well while other do not. The overall performance is 3.91 on 8 threads. The reason for the difference in the performance of the individual routines can be attributed to the granularity of the parallelized region. The table below shows the granularity and performance gain for each of 7 of the major routines.

	buts	jacu	blts	jacld	Rhs_303	Rhs_153	Rhs_56
Granularity	.02	.02	.02	.01	1.78	1.45	.97
Performance	3.6	2.45	4.67	4.15	7	1.69	7.03

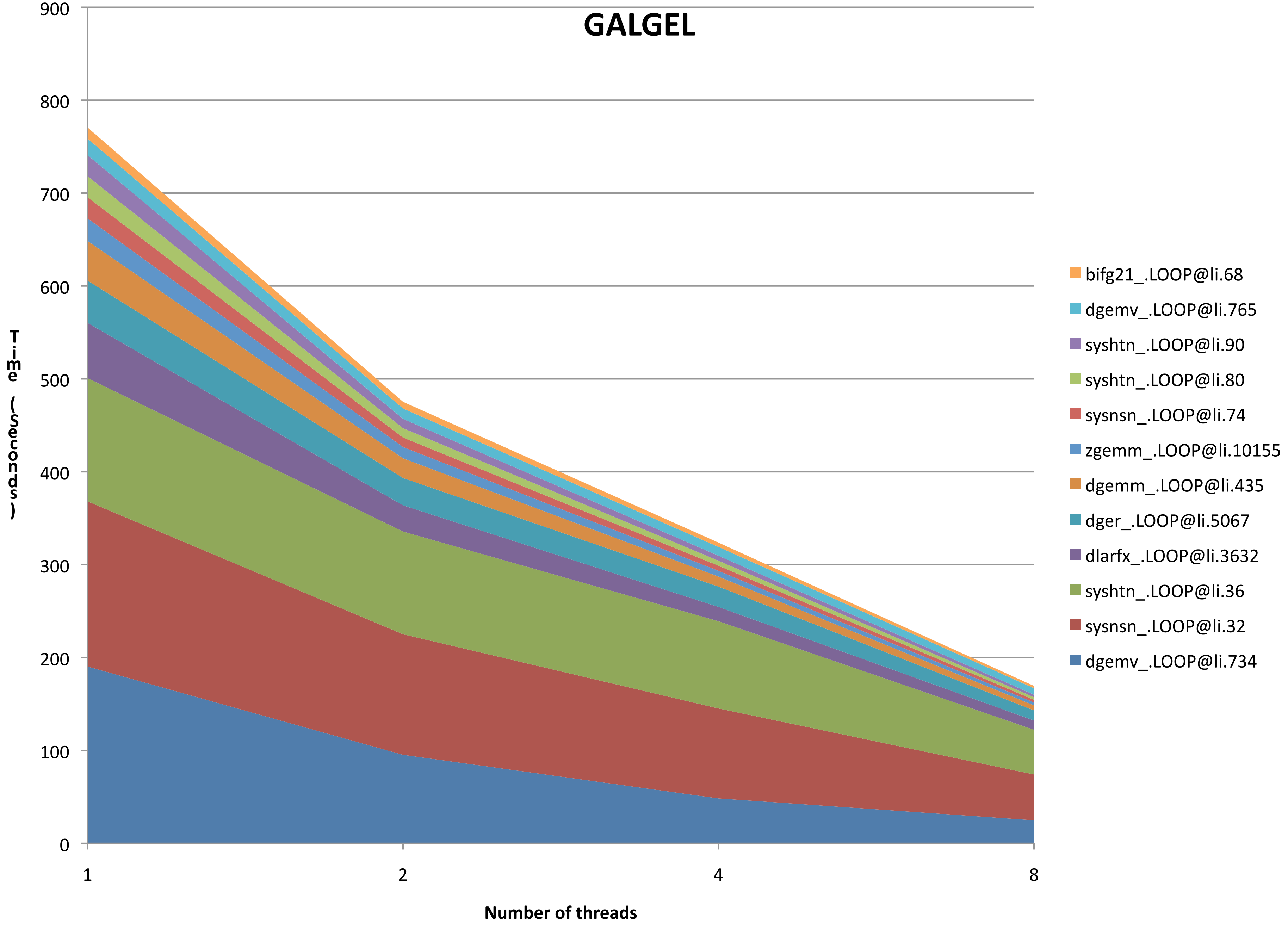
While it is not a linear function of granularity, the performance is highly dependent upon the granularity of the loop. In this case granularity is the time the loop takes to execute divided by the number of times the loop is executed. The other variation in the table could be due to the memory bandwidth requirement of the individual loops.



APPLU
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
0
2
4
6
8
10
12
14
Routines
granularity
performance on 8 threads

Performance is So-so

- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is not good
- Load Balance of the computation is good
- The computation that is parallelized is memory bandwidth limited



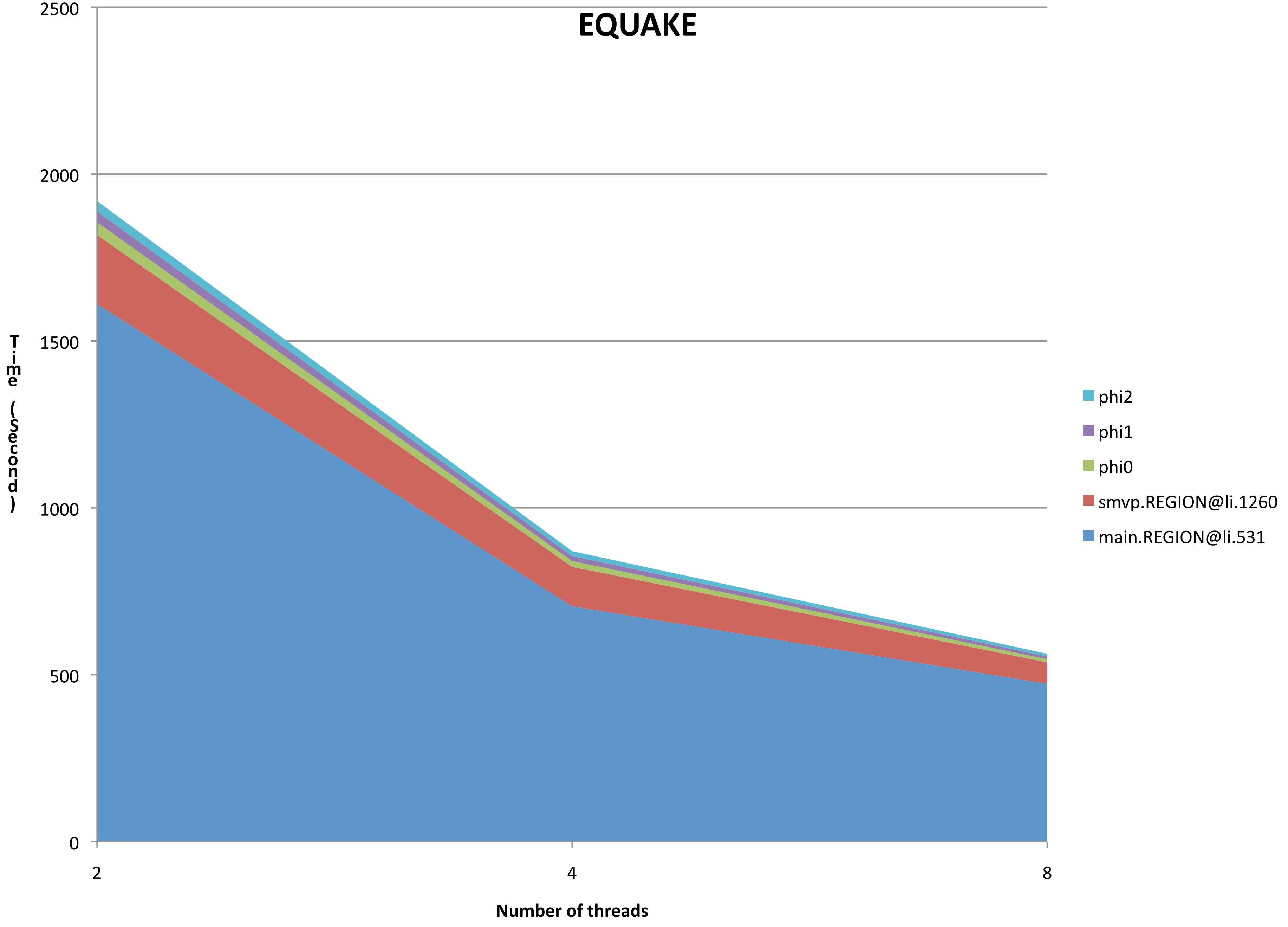
GALGEL
900
800
700
600
500
400
300
200
100
0
Time (Seconds)
1
2
4
8
Number of threads
bifg21_.LOOP@li.68
dgemv_.LOOP@li.765
syshtn_.LOOP@li.90
syshtn_.LOOP@li.80
sysnsn_.LOOP@li.74
zgemm_.LOOP@li.10155
dgemm_.LOOP@li.435
dger_.LOOP@li.5067
dlarfx_.LOOP@li.3632
syshtn_.LOOP@li.36
sysnsn_.LOOP@li.32
dgemv_.LOOP@li.734

Major Loop in GALGEL

```
!$OMP DO SCHEDULE(GUIDED)
                Ext12: Do LM = 1, K
                 L = (LM - 1) / NKY + 1
                 M = LM - (L - 1) * NKY
                  Do IL=1,NX
                   Do JL=1,NY
                    Do i=1,NKX
                     Do j=1,NKY
                        LPOP( NKY*(i-1)+j, NY*(IL-1)+JL ) = &
                                  WXTX(IL,i,L) * WXTY(JL,j,M) + &
                                  WYTX(IL,i,L) * WYTY(JL,j,M)
                     End Do
                    End Do
                   End Do
                  End Do
                  LPOP1(1:K) = MATMUL( LPOP(1:K,1:N), Y(K+1:K+N) )
                  Poj3( NKY*(L-1)+M, 1:K) = LPOP1(1:K)
                  Xp(NKY*(L-1)+M) =  DOT_PRODUCT (Y(1:K), LPOP1(1:K) )
                  Poj4( NKY*(L-1)+M,1:N) = &
                  MATMUL( TRANSPOSE( LPOP(1:K,1:N) ), Y(1:K) )
                 End Do Ext12
!$OMP END DO
```

Performance is So-so

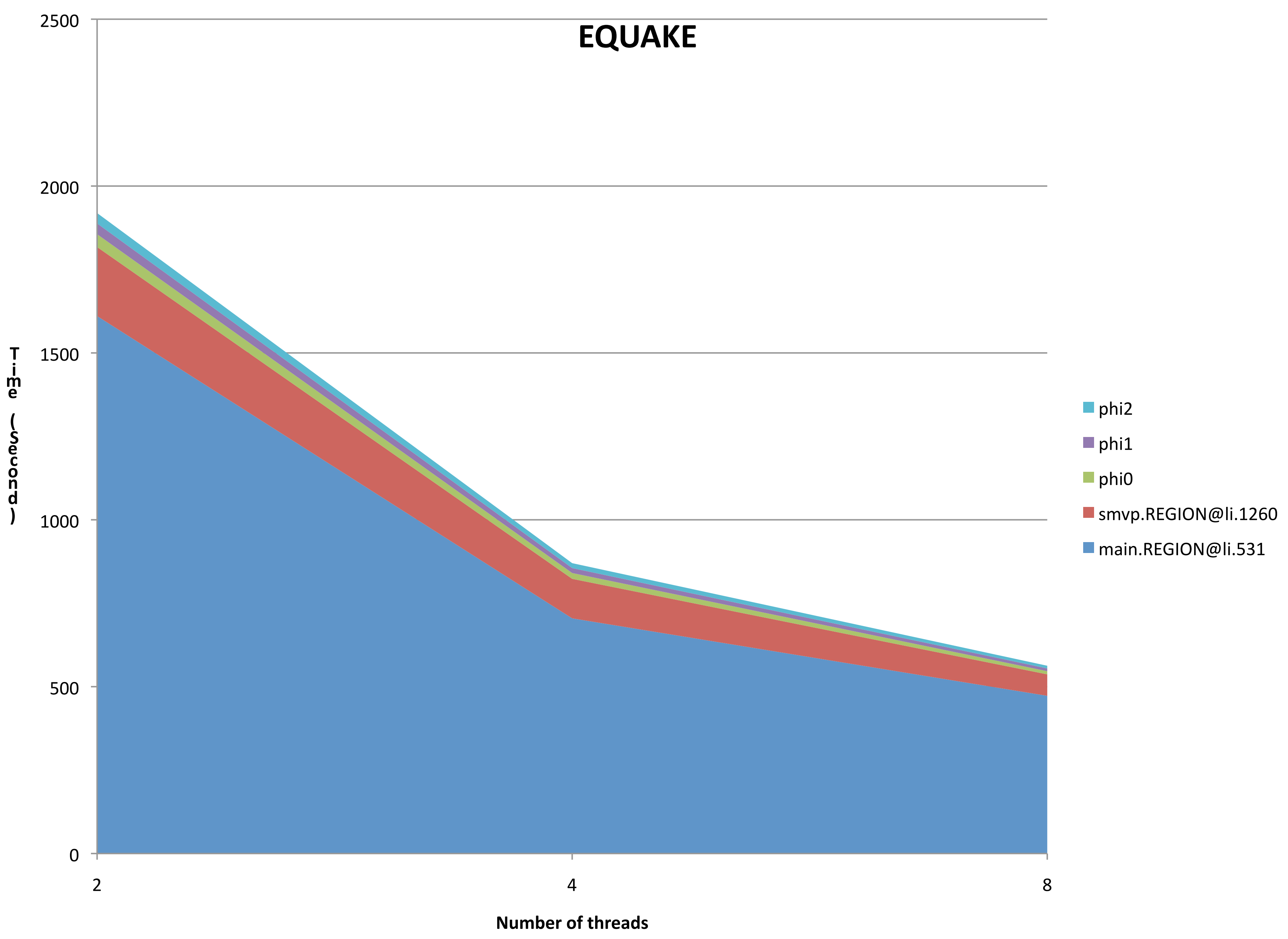
- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is not good
- Load Balance of the computation is good
- The computation that is parallelized is very memory bandwidth limited

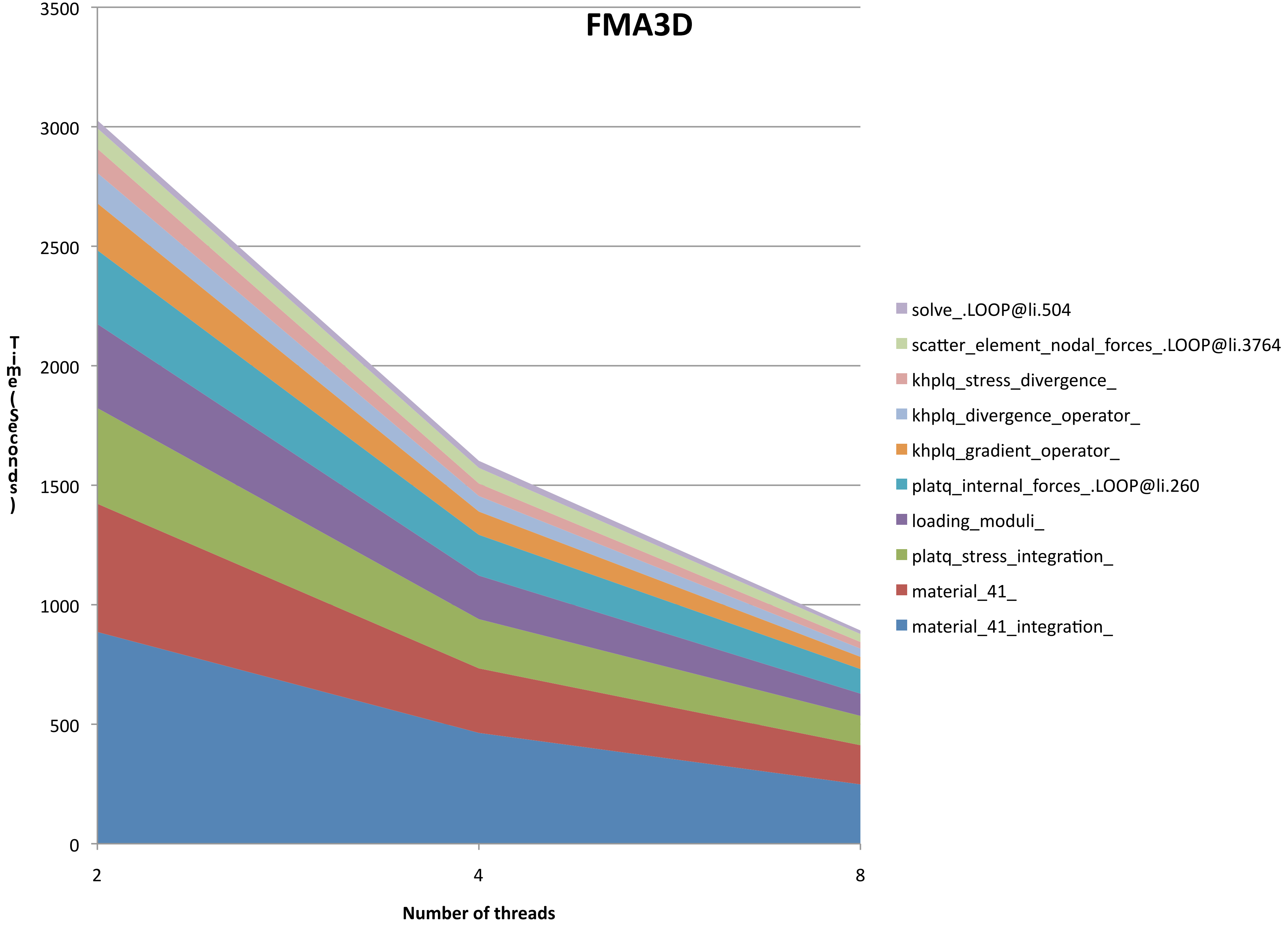

EQUAKE
2500
2000
1500
1000
500
0
Time (Second)
2
4
8
Number of threads
phi2
phi1
phi0
smvp.REGION@li.1260
main.REGION@li.531

```
#pragma omp for
  for (i = 0; i < nodes; i++) {
    Anext = Aindex[i];
    Alast = Aindex[i + 1];

    sum0 = A[Anext][0][0]*v[i][0] + A[Anext][0][1]*v[i][1] + A[Anext][0][2]*v[i][2];
    sum1 = A[Anext][1][0]*v[i][0] + A[Anext][1][1]*v[i][1] + A[Anext][1][2]*v[i][2];
    sum2 = A[Anext][2][0]*v[i][0] + A[Anext][2][1]*v[i][1] + A[Anext][2][2]*v[i][2];

    Anext++;
    while (Anext < Alast) {
      col = Acol[Anext];

      sum0 += A[Anext][0][0]*v[col][0] + A[Anext][0][1]*v[col][1] +
       A[Anext][0][2]*v[col][2];
      sum1 += A[Anext][1][0]*v[col][0] + A[Anext][1][1]*v[col][1] +
      A[Anext][1][2]*v[col][2];
      sum2 += A[Anext][2][0]*v[col][0] + A[Anext][2][1]*v[col][1] +
      A[Anext][2][2]*v[col][2];

      if (w2[my_cpu_id][col] == 0) {
        w2[my_cpu_id][col] = 1;
        w1[my_cpu_id][col].first = 0.0;
        w1[my_cpu_id][col].second = 0.0;
        w1[my_cpu_id][col].third = 0.0;
      }

      w1[my_cpu_id][col].first += A[Anext][0][0]*v[i][0] + A[Anext][1][0]*v[i][1] +
      A[Anext][2][0]*v[i][2];
      w1[my_cpu_id][col].second += A[Anext][0][1]*v[i][0] + A[Anext][1][1]*v[i][1] +
      A[Anext][2][1]*v[i][2];
      w1[my_cpu_id][col].third += A[Anext][0][2]*v[i][0] + A[Anext][1][2]*v[i][1] +
      A[Anext][2][2]*v[i][2];
      Anext++;
    }

    if (w2[my_cpu_id][i] == 0) {
      w2[my_cpu_id][i] = 1;
      w1[my_cpu_id][i].first = 0.0;
      w1[my_cpu_id][i].second = 0.0;
      w1[my_cpu_id][i].third = 0.0;
    }

    w1[my_cpu_id][i].first += sum0;
    w1[my_cpu_id][i].second += sum1;
    w1[my_cpu_id][i].third += sum2;
  }
}
```

Performance is Good

- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is questionable
- Load Balance of the computation is good
- The computation that is parallelized is not memory bandwidth limited


EQUAKE
2500
2000
1500
1000
500
0
Time (Second)
2
4
8
Number of threads
phi2
phi1
phi0
smvp.REGION@li.1260
main.REGION@li.531


FMA3D
3500
3000
2500
2000
1500
1000
500
0
Time (Seconds)
2
4
8
Number of threads
solve_.LOOP@li.504
scatter_element_nodal_forces_.LOOP@li.3764
khplq_stress_divergence_
khplq_divergence_operator_
khplq_gradient_operator_
platq_internal_forces_.LOOP@li.260
loading_moduli_
platq_stress_integration_
material_41_
material_41_integration_

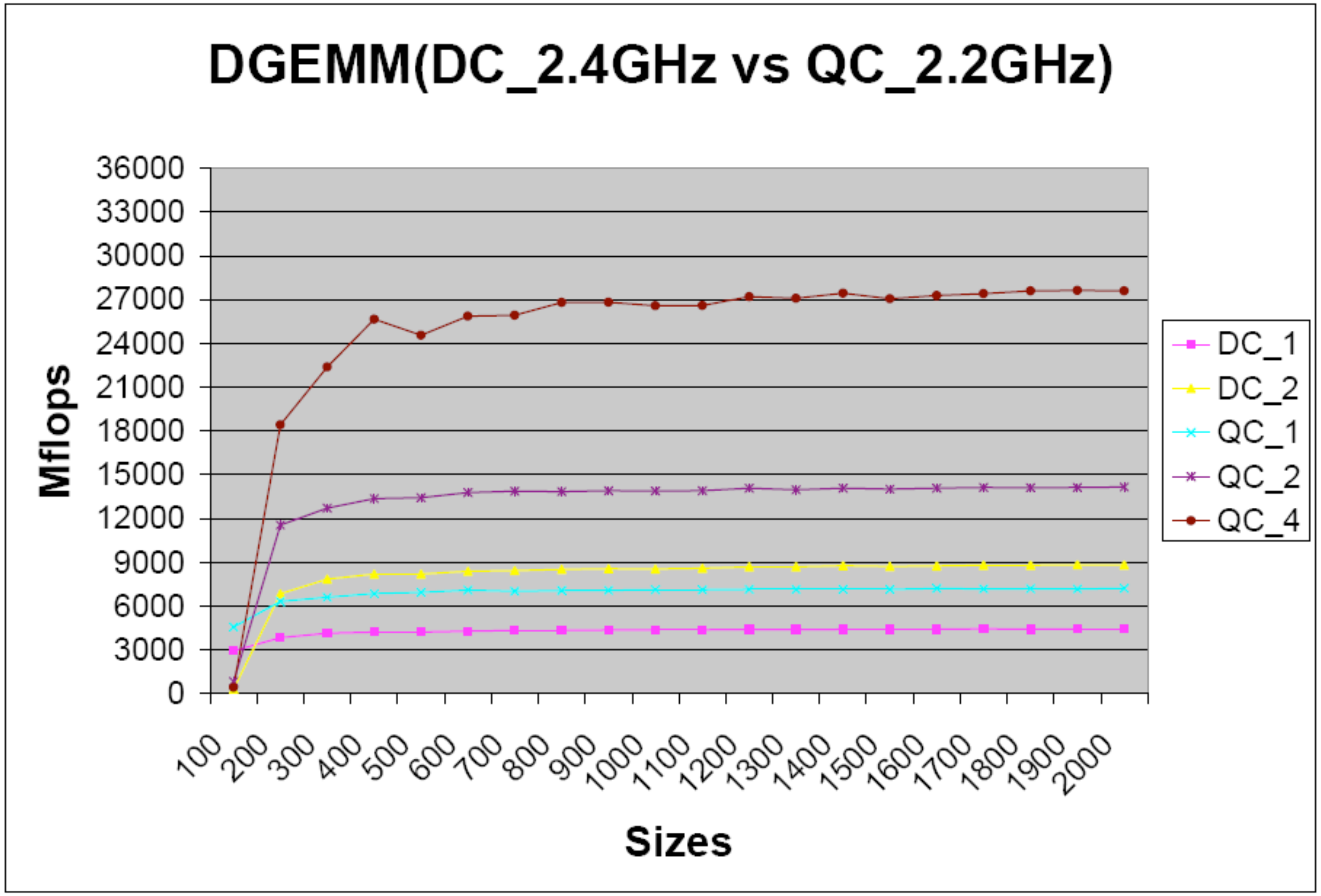
```
!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(N,IX)
        DO N = 1,NUMP4
          IF (PLATQ(N)%PAR%IGR .EQ. CURRENT_GROUP) THEN
            IX(1:8) = PLATQ(N)%PAR%IX(1:8)
            FORCE(IX(1:8))%Xint = FORCE(IX(1:8))%Xint + PLATQ(N)%RES%Xint
            FORCE(IX(1:8))%Yint = FORCE(IX(1:8))%Yint + PLATQ(N)%RES%Yint
            FORCE(IX(1:8))%Zint = FORCE(IX(1:8))%Zint + PLATQ(N)%RES%Zint
          ENDIF
        ENDDO
!$OMP END PARALLEL DO
```

Indirect address causing poor memory utilization – only achieve 2.5 in going from 2 to 8 threads

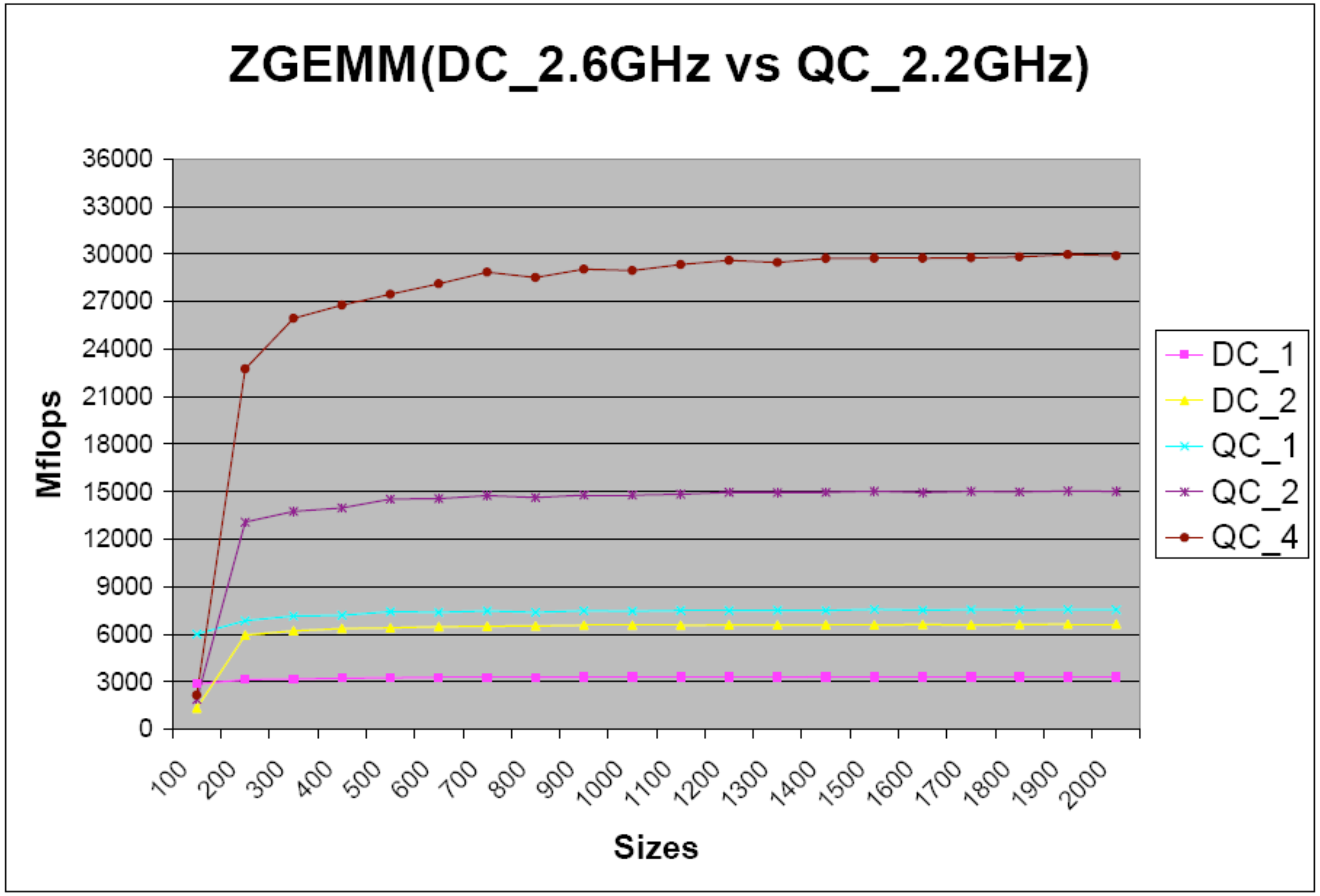
Performance is Good

- Large Percentage of the code that uses the computation time is parallelized
- Granularity of the computation is good
- Load Balance of the computation is good
- The computation that is parallelized is memory bandwidth limited

OpenMP in LIBSCI



OpenMP in LIBSCI


ZGEMM(DC_2.6GHz vs QC_2.2GHz)
Mflops
Sizes
DC_1
DC_2
QC_1
QC_2
QC_4